**Lab 4: Function Generator**

Authors: Benjamin Fields and Miles Alderman

ME 305-03

Dr. Espinoza-Wade

Cal Poly University

Spring 2022

From: Benjamin Fields and Miles Alderman

To: Dr. Espinosa-Wade

Date: May 26, 2022

Re: Lab 4 Report

## Overview

The objective of this lab was to implement a wave generator of variable period that can display a sawtooth, 7 or 15 segment sine, or square wave. This was done by using a periodically scheduled interrupt to incrementally change the output of a digital to analog converter, broken down into small step changes. These options are displayed to the user on an LCD user interface for their input. Additionally, the interface restricts user to input a wave period between 1 and 255 and returns errors for values that are out of bounds.

## Tasks

**Task 1: timer\_channel\_0**:

Initialization and activation/deactivation finite state machine for generating timer channel 0 interrupts. Task begins by enabling timer channel 0 interrupts, generating the first interrupt to offset the timer count, then waiting to turn on or turn off interrupts based on the status of RUN\_FLG.

States:

* timerstate0: initialization state. Set timer channel 0 for output compare, set channel to toggle, clear timer output compare flag, clear I bit, enable timer overflow flag and set debugger to freeze when execution ended. Next trigger one interrupt to acquire timing offset for execution of interrupt service routine and deactivate interrupts.
* timerstate1: Waiting to turn on interrupts. Test RUN\_FLG, if true, enable timer overflow flag and set output to toggle, move to timerstate2 on next passthrough.
* timerstate2: Waiting to turn off interrupts. Test RUN\_FLG, if false, disable timer overflow flag and clear toggle output, move to timerstate1 on next passthrough.

**Task 2: MASTERMIND**:

Master logic code for processing digit presses from the keypad, manipulation of display state flags, and manipulation of different variables used by the function generator. Upon keypress, mastermind decodes it, manages the logic for backend functions such as store nint values and wave selection, and checks for errors. The task also manipulates flags such as DWAVE and DPROMPT as opposed to that occurring in the function generator task.

States:

* masterstate0: initialization state. Sets MS\_WAVE\_FLG high at first to set initial post decoding.
* masterstate1: waiting for key state. Waits for key press, at which point it moves into decode state.
* masterstate2: decode state. The state first tests if the keypress was an enter or backspace key, in which case it goes to corresponding state. It then tests if the keypress was a digit by seeing if its ascii value is between $30 and $39. If it was a digit and MS\_WAVE\_FLG is high then mastermind will go to nint state on next passthrough. If digit but MS\_WAVE\_FLG is low, a wave needs to be selected and mastermind goes to Wave Key State. If none of these were selected the digit is ignored.
* masterstate3: Wave Key State. The state first tests if the digit is a zero, in which case the wave is being reset and the DMESS\_RESET flag is raised. If the digit is between 1 and 4, because those are the only keys we have digits defined for, and otherwise it is ignored. Finally, the DWAVE and NEW\_BTI flags are raised and the MS\_WAVE\_FLG is lowered.
* masterstate4: NINT key state. For digits being entered to NINT state, it first checks that there is space (less than 3 digits in buffer) and stores the key press if so.
* masterstate5: Backspace state. This state checks if the backspace is a valid entry under two conditions: that a function key is active and that there is indeed a digit to backspace. If so, it decrements the pointer to the key buffer and goes back to waiting. It also raises the DMESS\_BS flag for the display states.
* masterstate6: Enter key state. The state initially checks that it is accepting waves (MS\_WAVE\_FLG should be high). It then checks if it is an appropriate time to press enter by ensuring the DPROMPT flag is high. After, the zero count error is checked by making sure KEY\_COUNT is more than 0. At this point, the key buffer is ascii decoded and zero and magnitude errors are tested for. At this point, the enter keypress is valid. The DMESS\_ENT (to display enter state), NINT\_OK flag (to show it is ok to start generating), MS\_WAVE\_FLG (to start accepting new wave numbers) and all set high. The KEY\_COUNT and DPROMPT are cleared. The result of decoding is then stored in NINT for the number of interrupts per cycle, and the task returns to waiting for key press.
* masterstate7: Error key state. This state is for if there is an error set and an error message is being set. It compares if the error count is the same as the max value of error count, in which case the message can be cleared.

**Task 3: DISPLAY\_TASK**:

I/O interfacing code to display the wave selection, prompt user input, and display errors as determined by mastermind. DISPLAY\_TASK does not contain any conditional logic and relies on prompts from mastermind to function. DISPLAY\_TASK is either actively printing a single character per passthrough, determining the next message to display, or holding in an error pattern.

States:

The following states are different instances of messages display needs to print, determined by display hub.

* displaystateinit0: Initialization. Set error count 1 and 2 to maximum, set FIRSTCH flag to allow initialization printing, move to displaystateinit1 on next passthrough.
* displaystateinit1: Line 1 Initialization. Print the default message prompting the user to input a wave numbered between 1 and 4.
* displaystatehub: Central Display Hub. Test all DMESS flags generated by mastermind for a new message to print, otherwise exit task.
* displaystateWAVE: Display selected wave and prompt user for NINT input. When DWAVE is true, determine which wave value has been selected and begin printing wave selection and NINT prompt on second line. Display continues printing full message one character at a time every passthrough, once finished DWAVE is cleared, cursor is hidden, and return to displaystatehub.
* displaystateNINT: Echoing user input digits to NINT user prompt on display. When DPROMPT is true, generated by mastermind, character is loaded via indexed addressing and stored in ECHO, and subsequently printed at proper cursor address and cursor address is incremented.
* displaystateBS: Backspace. When DMESS\_BS is true, state moves the cursor back 1 space and prints an ASCII $20, or a space to that cursor location. Printing logic flags are cleared and DMESS\_BS is cleared prior to returning to displaystatehub.
* displaystateENT: Enter. Moves cursor to hide under 1 on line 1 when DMESS\_ENT is true. DMESS\_ENT is cleared prior to returning to displaystatehub.
* displaystateEB: NINT input too big. Print overflow error in place of NINT user prompt when DMESS\_EB is true. Once done printing message, move to errordelay on next passthrough and hide cursor and clear DMESS\_EB.
* displaystateEZ: NINT input equals zero. Print zero error in place of NINT user prompt when DMESS\_EZ is true. Once done printing message, move to errordelay on next passthrough and hide cursor and clear DMESS\_EZ.
* displaystateEN: NINT has no input. Print no digits entered error in place of NINT user prompt when DMESS\_EN is true. Once done printing message, move to errordelay on next passthrough and hide cursor and clear DMESS\_EN.
* errordelay: Delay loop for erroneous user input. Decrement ERRORCOUNT by one every passthrough until it equals zero. Once equal to zero, reset ERRORCOUNT value to maximum and move to displaystatekeyreset on next passthrough.
* displaystatekeyreset: Error message clearance and NINT prompt reprint. Once error delay has run its course, load NINT user prompt and print one character every passthrough. Once finished, move cursor to first NINT digit location and return to displaystatehub.
* displaystatereset: Bottom line screen clear. When DMESS\_RESET is true, print a line of ASCII spaces to the entire bottom line, reset printing logic and DMESS\_RESET, then return to displaystatehub.

**Task 4: KEYPAD\_TASK**:

I/O interfacing code to store ASCII values of characters pressed by the user on the keypad. Task notifies mastermind to collect key input value and waits for mastermind acknowledgement to allow for additional keypresses to be stored.

States:

* keystate0: Initialize keystate, move to state 1 on next passthrough.
* keystate1: Test LKEY\_FLG to see if key has been pressed, if so, set KEY\_FLG, retrieve character and store in KEY\_BUFFER, then move to keystate2 on next passthrough. If no key pressed, ignore and stay in keystate1.
* keystate2: Test KEY\_FLG for mastermind key retrieval acknowledgement. If KEY\_FLG cleared, key has been registered and move to keystate1 on next passthrough to continue gathering key entries.

**Task 5: function\_generator**:

Initialization and activation/deactivation finite state machine for generating timer channel 0 interrupts. Task begins by setting timer channel 0 for output compare, set timer channel 0 to toggle on interrupt, clears output compare flag,

States

* fxngenstate0: initialization state
* fxngenstate1: waiting for wave state. Tests if there is wave and then waits for wave to be selected. It stores address for start of waveform generation information in wave pointer.
* fxngenstate2: New wave state. Test if display of wave message is done being displayed to user, indicated by DWAVE. Once printed, it then fills wave generation variables with info from wave selected, then loads pointer to the wave segment pieces. At this point, the only missing information for new wave generation is the number of interrupts per BTI, NINT.
* fxngenstate3: waiting for NINT state. State is waits for user input of nint value and checks to make sure it is valid. The first segment (LSEG) of wave generated needs to be incremented manually because it isn’t incremented in the wave generation loop on the first pass. The state also puts 0 it NINT\_OK to move to state 4 and raises the RUN\_FLG.
* fxngenstate4: display wave state. The state first checks to make sure a wave isn’t already being generated. If it is at the start of a NEW\_BTI, the state is exited since we don’t want to change any of the wave generation info yet. If it is at the start of a new BTI, it decrements the segment counter and adds the new DAC value for the next BTI. If the number of segments left, CSEG, was not zero, the state loads the information for the next segment. Otherwise, the generator has finished a period of the wave generation and the WAVEPTR is reinitialized to start a new period.

**Inter-Task Communication Variables**

The next two pages contain a table of our inter-task communication variables.

|  |  |  |  |
| --- | --- | --- | --- |
| Communication Variable | Description | Variable Cleared | Variable Set |
| KEY\_FLG | Flag mirroring LKEY\_FLG generated when key is pressed | Task\_2: Mastermind; masterstate2, masterstate3, masterstate4, masterstate5, masterstate6 | Task\_4: Keypad; keystate1 |
| DMESS\_BS | Flag telling display to print a space in previous digit location | Task\_3: Display; displaystateBS | Task\_2:Mastermind; masterstate5 |
| DMESS\_ENT | Flag telling display to hide cursor | Task\_3: Display; displaystateENT | Task\_2:Mastermind; masterstate6 |
| DMESS\_EB | Flag telling display to print overflow error in top line | Task\_3: Display; displaystateEB | Task\_2:Mastermind; masterstate6 |
| DMESS\_EZ | Flag telling display to print zero error in top line | Task\_3: Display; displaystateEZ | Task\_2:Mastermind; masterstate6 |
| DMESS\_EN | Flag telling display to print no digits entered error in top line | Task\_3: Display; displaystateEN | Task\_2:Mastermind; masterstate6 |
| DMESS\_RESET | Flag telling display to fully reset the screen | Task\_3: Display; displaystatereset | Task\_2: Mastermind; masterstate3 |
| ERRORCOUNT | Remaining passes through main before clearing error message in line 1 | Task\_3: Display; errordelay1 | Task\_3: Display; errordelay1 |
| KEY\_COUNT | Variable counting the number of keys stored in buffer | Task\_2: Mastermind; masterstate5, masterstate 6 | Task\_2: Mastermind; masterstate3 |
| WAVE | Variable storing the current wave type to be generated | Task\_2: Mastermind; masterstate3 | Task\_2: Mastermind; masterstate3 |
| DWAVE | Flag telling display to print the description and NINT prompt for the number contained in WAVE | Task\_3: Display; displaystateWAVE | Task\_2: Mastermind; masterstate3 |
| DPROMPT | Flag telling display to print the NINT digit entered | Task\_2: Mastermind; masterstate6 Task\_3: Display; displaystateNINT | Task\_2: Mastermind; masterstate4 |
| NINT\_OK | Flag telling function generator to begin generating the waveform | Task\_5: Function\_generator; fxngenstate3 | Task\_2: Mastermind; masterstate6 |
| RUN\_FLG | Boolean for activating or deactivating timer channel 0 interrupts | Task\_2: Mastermind; masterstate3 | Task\_5: Function\_generator; fxngenstate3 |
| NINT | Variable storing the count of interrupts per BTI | Task\_2: Mastermind; masterstate6 | Task\_2: Mastermind; masterstate6 |
| NEW\_BTI | Boolean for indicating if function generator should move to a new BTI | Task\_5: Function\_generator; fxngenstate4 | Task\_2: Mastermind; masterstate3, ISR |

**Finite State Machine Diagrams:**

The following figures contain the finite state machine diagrams for Task 1-8:

Diagram

Description automatically generated

Figure 1: Timer Channel Zero FSM

Diagram

Description automatically generated

Figure 2: Mastermind FSM

Diagram

Description automatically generated

Figure 3. Task\_3: Display Finite State Machine Diagram

Diagram

Description automatically generated

Figure 4. Task\_2: Keypad Finite State Machine Diagram.

Diagram

Description automatically generated

Figure 5: Function Generator FSM

### Source Code: See addended .pdf